Loss functions and learning algorithms for neural network

Rafael Baeta

Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

26 de maio de 2017

Introduction

- We often use neural networks as a black box
- Some parts of the Neural network receive less importance
- A complete neural network architecture can be, in a simplified way, composed of:

- Layers (Convolution, Max Pooling, Fully Connected, etc)
- Learning algorithm (Gradient descent, Newton's method, etc)
- Loss function (NLL, Cross Entropy, Quadratic Cost, etc)
- Classifier (softmax, SVM, etc)

Introduction

- Normally we use a know architeture

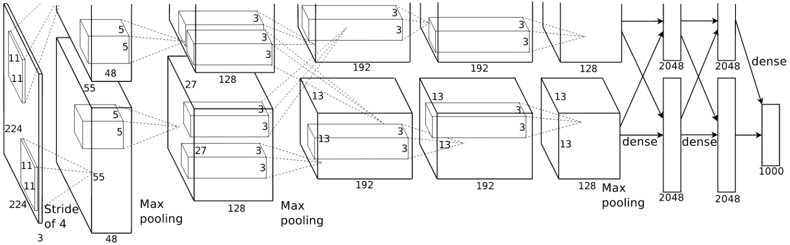


Figura : AlexNet

1

---

[1] https://kratzert.github.io/2017/02/24/finetuning-alexnet-with-tensorflow.html

Introduction

- Define default parameters (AlexNet)

- Learning rate: 0.01
- Weight decay: 0.0005
- Momentum: 0.9
- Learning algorithm: SGD

Introduction

- if acc > 90%
    - Right. Let's write a paper

Introduction

- if acc > 90%
  - Right. Let's write a paper
- else
  - Cry and try to change number of layers, neurons, parameters and lastly data.

Introduction

But...

Introduction

But...

And the loss function and learning algorithm?

Introduction

But...

What about the loss function and the learning algorithm?

They don't change anything?

Loss functions

- A loss function measures the model error
- It is used whenever the model is updated
- The aim of the learning algorithm is to minimize this function

Loss functions

Example:

Suppose we have an advertisement on a website:

- We have a bid and revenue for click.
- We want to maximize our revenue by accurately predicting a click
- And we want to minimize our cost

So, we have:

$$L(p) = \sum_i b * p_i - (r_c * y_i) p_i \quad (1)$$

where: b = fixed bid, p_i = prediction, r_c = revenue and y_i = right prediction

Loss functions

In this equation we have 2 parts:

- A penalty when you miss a prediction.
- And a cost minization part

So, we have:

$$L(p) = \sum_i b \text{ (penalty) } * p_i - (r_c * y_i) p_i \text{ (cost minimization)}$$

where:

b = fixed bid, p_i = prediction, r_c = revenue and y_i = right prediction

Loss functions

In our example supposse a prediction vector (p_i) and groundtruth vector (y_i).

In a ideal case we have all $p_i == y_i$, so, for $b = 10$, $r_c = 20$ and $n = 4$:

$$L(p) = \sum_i b * p_i \text{ (penalty) } -(r_c * y_i) p_i \text{ (cost minimization)}$$

$$L(p) = 4 * (10 - 20) = 4 * (-10) = -40$$

The worse case is when all $p_i! = y_i$ and p have only one, in this case, we have:

$$L(p) = 4 * 10 = 40$$

Loss functions

For an $p_i = 0, 1, 1, 0, 1$ and $y_i = 1, 1, 1, 0, 0$ we have:

$$L(p) = \sum_i b * p_i \text{ (penalty) } -(r_c * y_i)p_i \text{ (cost minimization)}$$

$$L(p) = 10 * 0 - (20 * 1) * 0 + \tag{2}$$

Loss functions

For an $p_i = 0, 1, 1, 0, 1$ and $y_i = 1, 1, 1, 0, 0$ we have:

$$L(p) = \sum_i b * p_i \text{ (penalty) } - (r_c * y_i) p_i \text{ (cost minimization)}$$

$$\begin{aligned} L(p) = {} & 10 * 0 - (20 * 1) * 0 + \\ & 2 * (10 * 1 - (20 * 1) * 1) + \end{aligned} \tag{3}$$

Loss functions

For an $p_i = 0, 1, 1, 0, 1$ and $y_i = 1, 1, 1, 0, 0$ we have:

$$L(p) = \sum_i b * p_i \text{ (penalty) } -(r_c * y_i)p_i \text{ (cost minimization)}$$

$$\begin{aligned} L(p) = {} & 10 * 0 - (20 * 1) * 0 + \\ & 2 * (10 * 1 - (20 * 1) * 1) + \\ & 10 * 0 - (20 * 1) * 0 + \end{aligned} \tag{4}$$

Loss functions

For an $p_i = 0, 1, 1, 0, 1$ and $y_i = 1, 1, 1, 0, 0$ we have:

$$L(p) = \sum_i b * p_i \text{ (penalty) } -(r_c * y_i) p_i \text{ (cost minimization)}$$

$$\begin{aligned} L(p) = {} & 10 * 0 - (20 * 1) * 0 + \\ & 2 * (10 * 1 - (20 * 1) * 1) + \\ & 10 * 0 - (20 * 1) * 0 + \\ & 10 * 1 - (20 * 0) * 1 = -10 \end{aligned} \tag{5}$$

Loss functions

symbol	name	equation
$\mathscr{L}_1$	L_1 loss	$\|\|y-o\|\|_1$
$\mathscr{L}_2$	L_2 loss	$\|\|y-o\|\|_2^2$
$\mathscr{L}_1 \circ \sigma$	expectation loss	$\|\|y-\sigma(o)\|\|_1$
$\mathscr{L}_2 \circ \sigma$	regularised expectation loss	$\|\|y-\sigma(o)\|\|_2^2$
$\mathscr{L}_\infty \circ \sigma$	Chebyshev loss	$max_j \|\sigma(o)^j - y^{(j)}\|$
log	log (cross entropy) loss	$-\sum_j y^{(j)} log \sigma(o)^j$
log^2	squared log loss	$-\sum_j [y^{(j)} log \sigma(o)^j]^2$
hinge	hinge (margin) loss	$\sum_j max(0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)})$
$hinge^2$	squared hinge (margin) loss	$\sum_j max(0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)})^2$
$hinge^3$	cubed hinge (margin) loss	$\sum_j max(0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)})^3$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(o)^{(j)} y^{(j)}}{\|\|\sigma(o)\|\|_2^2 + \|\|y\|\|_2^2 - \sum_j \sigma(o)^{(j)} y^{(j)}}$
D_{cs}	Cauchy-Schwarz Divergence	$-log \frac{\sum_j \sigma(o)^{(j)} y^{(j)}}{\|\|\sigma(o)\|\|_2 \|\|y\|\|_2}$

where **y** is true label as one-hot coding, $\hat{y}$ is true label as **+1/-1**, **o** is the output of the last layer, $\sigma(.)$ denotes probability

Loss functions - $\mathscr{L}_p$

- $\mathscr{L}_p$ is considered regressive losses
- $\mathscr{L}_p$ applied to the probability leads to minimization of expected misclassification probability ($\mathscr{L}_p \circ \sigma$)
- This property become $\mathscr{L}_p \circ \sigma$ robust to outliers/noise
- But, these loss functions are not popular? Why?

Loss functions - $\mathscr{L}_p$

- It don't have monotonic partial derivatives
- Because of this, learning becomes slow in heavily misclassified examples

Proof:

$$C = \frac{(y-o)^2}{2} \tag{6}$$

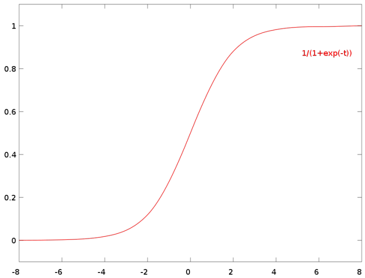
where a = $\sigma(z)$, where $z = wx + b$ and σ are the sigmoid function

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z) \tag{7}$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z)x = a\sigma'(z) \tag{8}$$

Loss functions - $\mathscr{L}_p$

- Let's look for the shape of σ function



$$\frac{\partial C}{\partial w} = a\sigma'(z) \text{ and } \frac{\partial C}{\partial b} = a\sigma'(z)$$

2

[2] https://en.wikipedia.org/wiki/Talk%3ASigmoid_function

Loss functions - Cross Entropy

- Log loss function (Cross entropy) minimize the same way
- But, this function is not affected by slow learning
- This is because $\sigma'(z)$ is eliminated in the cost equation
- Therefore, it correctly penalizes heavily misclassified examples

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y) \text{ and } \frac{\partial C}{\partial b_j} = \frac{1}{n}\sum_x(\sigma(z) - y)$$

Loss functions - Hinge Loss

- Hinger loss is used for ”maximum-margin” classification
- Commonly used for support vector machines (SVM) for binary problems

Ex: For a output t = +/- 1 and a classifier score y the hinge loss of y is:

$$l(y) = max(0, 1 - t * y)$$

where y = w*x + b

Loss functions - Hinge Loss

- Hinger loss can be extended to the multiclass classification:

$$l(y) = max(0, 1 + \max_{t \neq y} W_t X - W_y X)^3$$

$$l(y) = \sum_{t \neq y} max(0, 1 + W_t X - W_y X)^4$$

[3] Koby Crammer e Yoram Singer. "On the algorithmic implementation of multiclass kernel-based vector machines". Em: *Journal of machine learning research* 2.Dec (2001), pp. 265–292.

[4] Urün Dogan, Tobias Glasmachers e Christian Igel. "A unified view on multi-class support vector classification". Em: *Journal of Machine Learning Research* 17.45 (2016), pp. 1–32.

Loss functions - Hinge Loss

- Hinger loss is a convex function, so convex optimizers in machine learning can work (SGD):
- But, it is not differentiable at **ty=1**!
- However, there exist subgradient and smoothed versions[5]:

$$l(y) = \begin{cases} \frac{1}{2} - ty, & \text{if } ty \leqslant 0 \\ \frac{1}{2} - (1 - ty)^2, & \text{if } 0 < ty \leqslant 1 \\ 0, & \text{if } 1 \leqslant ty \end{cases} \tag{9}$$

[5] Jason DM Rennie e Nathan Srebro. “Loss functions for preference levels: Regression with discrete ordered labels”. Em: *Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling*. Kluwer Norwell, MA. 2005, pp. 180–186.

Loss functions - High order

- High order for hinge losses help in speed and final performance
- This does not hold for higher order log losses
- And for $\mathscr{L}_p$ its help to reduce the high penalties

Loss functions

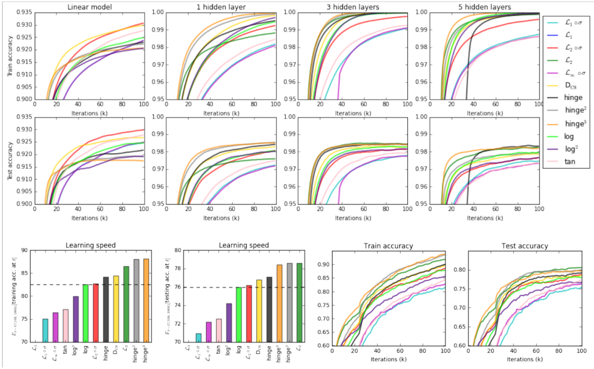


Figura : Train and test on MNIST and Cifar

6

[6] Katarzyna Janocha e Wojciech Marian Czarnecki. “On Loss Functions for Deep Neural Networks in Classification”. Em: *arXiv preprint arXiv:1702.05659* (2017).

Loss functions

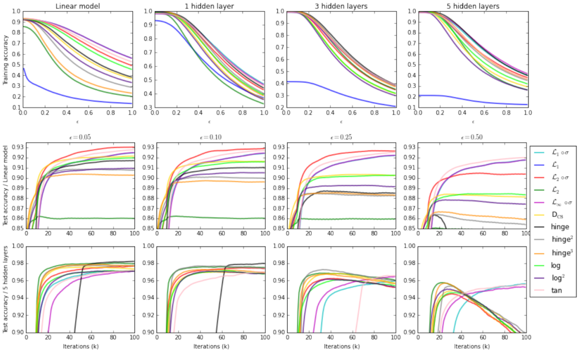


Figura : Robust for noise in MNIST

7

[7] Janocha e Czarnecki, "On Loss Functions for Deep Neural Networks in Classification".

Learning algorithm

A learn algorithm is used to teach the neural network. The most commonly used learning algorithm is **Gradient descent** and there are a some others:

- Newton's method
- Conjutage gradient
- Quasi Newton
- Levenberg Marquardt

Learning algorithm - Memory and speed comparison

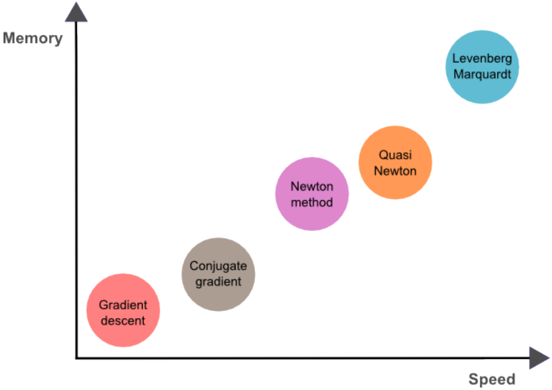


Figura : Comparison of optimization methods

8

[8] **https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network.**

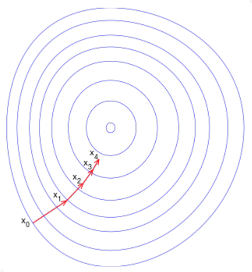
Learning algorithm - Gradient descent

There are three ways to use GD:

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Learning algorithm - Gradient descent

Gradient descent try to minimize a **loss function** $J(\theta)$:



The parameters are updated following the equation bellow:

$$\theta = \theta - \eta * \nabla_\theta J(\theta) \tag{10}$$

9

[9]https://stackoverflow.com/questions/35711315/gradient-descent-vs-stochastic-gradient-descent-algorithms

Learning algorithm - Gradient descent variations

There are some variations of gradient descent:

- Momentum
- Nesterov
- Adagrad
- Adadelta
- RMSprop
- Adam

Learning algorithm - Gradient descent variations - Momentum

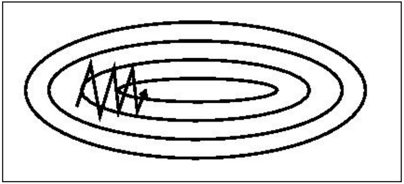
- SGD has trouble where the surface curves much more steeply in one dimension than in another
- Momentum helps accelerate SGD in relevant direction

$$\upsilon_t = \gamma \upsilon_{t-1} + \eta \nabla_\theta J(\theta)$$

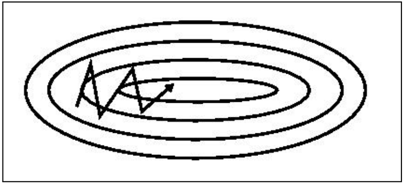
$$\theta = \theta - \upsilon_t$$

Learning algorithm - Gradient descent variations - Momentum

- Essentially, when using momentum, we push a ball down a hill.
- The ball accumulates momentum as it rolls downhill.



SGD without momentum



SGD with momentum

[10]

[10] http://sebastianruder.com/optimizing-gradient-descent/index.htm

Learning algorithm - Gradient descent variations - Nesterov

- But, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory.
- What if we had a ball that knows where it's going?

$$\upsilon_t = \gamma \upsilon_{t-1} + \eta \nabla_\theta J(\theta - \gamma \upsilon_{t-1})$$

$$\theta = \theta - \upsilon_t$$

Learning algorithm - Gradient descent variations - Nesterov

- Computing $\theta - \gamma \upsilon_{t-1}$ thus gives us an approximation of the next position of the parameters.
- We can now calculate the approximate future position of our parameters

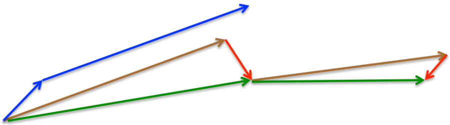


Figura : Nesterov update

11

[11] http://sebastianruder.com/optimizing-gradient-descent/index.htm

Learning algorithm - Gradient descent variations - Adagrad

- It's adapts the learning rate to the parameters.
- Larger updates for infrequent and smaller updates for frequent parameters

$$g_{t,i} = \nabla_\theta J(\theta_i)$$

$$\theta_{t+1,i} = \theta_{t,i} - \eta * g_{t,i}$$

Learning algorithm - Gradient descent variations - Adagrad

- Modifies the general learning rate ∇ at each step t for each parameter θ_i.
- It's done based on the past gradients that have been computed for θ_i

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_t + \varepsilon}}$$

Learning algorithm - Gradient descent variations - Adagrad

- The main benefits of Adagrad are that it eliminates the need to manually adjust the learning rate
- Its main weakness is its accumulation of square gradients in the denominator
- As each term added is positive, the accumulated sum continues to grow during training
- The learning rate eventually becomes infinitesimally small

Learning algorithm - Gradient descent variations - Adadelta

- Adadelta is an extension of Adagrad
- Its tries to reduce his aggressiveness
- Adadelta constrains the window of accumulated past gradients to some fixed size w
- Instead of inefficiently storing previous square gradients, the sum of the gradients is defined recursively as a decaying average of the entire past square gradient

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}}$$

Learning algorithm - Gradient descent variations - RMSProp and Adam

- Same objective of Adadelta
- Adam, RMSProp and Adadelta share the exponentially decreasing average of square gradients
- Adadelta has moving average biased by initialization of decay parameters
- Adam has bias correction

Conclusion

- We must pay attention to all parts of an architecture
- It is difficult to choose a loss function depending on the problem
- A loss function and a learning algorithm suitable for a problem can have a major impact on performance

References I

- *5 algorithms to train a neural network*. https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network
- Chen, Si e Yufei Wang. "Convolutional neural network and convex optimization". Em: *Dept. of Elect. and Comput. Eng., Univ. of California at San Diego, San Diego, CA, USA, Tech. Rep* (2014).
- Crammer, Koby e Yoram Singer. "On the algorithmic implementation of multiclass kernel-based vector machines". Em: *Journal of machine learning research* 2.Dec (2001), pp. 265–292.
- *CS231n Convolutional Neural Networks for Visual Recognition*. http://cs231n.github.io/neural-networks-3/.
- *CS231n Convolutional Neural Networks for Visual Recognition - Optimization*. http://cs231n.github.io/optimization-1/.

References II

- Dogan, Urün, Tobias Glasmachers e Christian Igel. “A unified view on multi-class support vector classification”. Em: *Journal of Machine Learning Research* 17.45 (2016), pp. 1–32.
- *How do you decide which loss function to use for machine learning?* https://www.quora.com/How-do-you-decide-which-loss-function-to-use-for-machine-learning.
- *Improving the way neural networks learn*. http://neuralnetworksanddeeplearning.com/chap3.html.
- Janocha, Katarzyna e Wojciech Marian Czarnecki. “On Loss Functions for Deep Neural Networks in Classification”. Em: *arXiv preprint arXiv:1702.05659* (2017).
- *Optimization for Deep Networks*. http://www.cs.cmu.edu/ imisra/data/Optimization_2015_11_11.pdf.

References III

Rennie, Jason DM e Nathan Srebro. “Loss functions for preference levels: Regression with discrete ordered labels”. Em: *Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling*. Kluwer Norwell, MA. 2005, pp. 180–186.

Ruder, Sebastian. “An overview of gradient descent optimization algorithms”. Em: *arXiv preprint arXiv:1609.04747* (2016).